The following is a full Nginx FastCGI cache configuration example making use of most of the cache-related directives described in the preceding table:

```
fastcgi_cache phpcache;
fastcgi_cache_key "$scheme$host$request_uri"; # $request_uri includes
the request arguments (such as /page.php?arg=value)
fastcgi_cache_min_uses 2; # after 2 hits, a request receives a cached
response
fastcgi_cache_path /tmp/cache levels=1:2 keys_zone=phpcache:10m
inactive=30m max_size=500M;
fastcgi_cache_use_stale updating timeout;
fastcgi_cache_valid 404 1m;
fastcgi_cache_valid 500 502 504 5m;
```

Since these directives are valid for pretty much any virtual host configuration, you may want to save these in a separate file (`fastcgi_cache`) that you can include at the appropriate place:

```
server {
    server_name website.com;
    location ~* \.php$ {
         fastcgi_pass 127.0.0.1:9000;
         fastcgi_param SCRIPT_FILENAME /home/website.com/www$fastcgi_
script_name;
         fastcgi_param PATH_INFO $fastcgi_script_name;
         include fastcgi_params;
         include fastcgi_cache;
    }
}
```

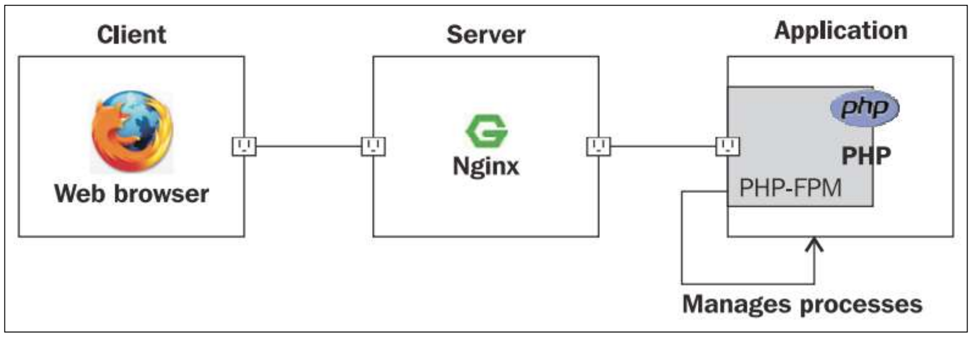
PHP with Nginx

We are now going to configure PHP to work together with Nginx via FastCGI. Why FastCGI in particular, as opposed to the other two alternatives SCGI and uWSGI? The answer came with the release of PHP version 5.3.3. As of this version, all releases come with an integrated FastCGI process manager allowing you to easily connect applications implementing the FastCGI protocol. The only requirement is that your PHP build should have been configured with the `--enable-fpm` argument. If you are unsure whether your current setup includes the necessary components, worry not: a section of this chapter is dedicated to building PHP with everything we need. Alternatively, the `php-fpm` or `php5-fpm` packages can be found in most repositories.

Architecture

Before starting the setup process, it's important to understand the way PHP interacts with Nginx. We have established that FastCGI is a communication protocol running through sockets, which implies that there is a client and a server. The client is obviously Nginx. As for the server, well, the answer is actually more complicated than just "PHP".

By default, PHP supports the FastCGI protocol. The PHP binary processes scripts and is able to interact with Nginx via sockets. However, we are going to use an additional component to improve the overall process management: the FastCGI Process Manager, also known as *PHP-FPM*.



PHP-FPM takes FastCGI support to an entirely new level. Its numerous features are detailed in the next section.

PHP-FPM

The process manager, as its name suggests, is a script that manages the PHP processes. It awaits and receives instructions from Nginx, and runs the requested PHP scripts under the environment that you configure. In practice, PHP-FPM introduces a number of possibilities such as:

- Automatically *daemonizing* PHP (turning it into a background process)
- Executing scripts in a *chrooted* environment
- Improved logging, IP address restrictions, pool separation, and much more

Setting up PHP and PHP-FPM

In this section, we will detail the process of downloading and compiling a recent version of PHP. You will need to go through this particular step if you are currently running an earlier version of PHP (<5.3.3).

Downloading and extracting

At the time of writing, the latest stable version of PHP is 5.6.10. Download the tar ball via the following command:

```
[user@local ~]$ wget http://php.net/get/php-5.6.10.tar.gz/from/www.php.
net/mirror
```

Once downloaded, extract the PHP archive with the `tar` command:

```
[user@local ~]$ tar xzf php-5.6.10.tar.gz
```

Requirements

There are two main requirements for building PHP with PHP-FPM: the `libevent` and `libxml` development libraries. If these are not already installed on your system, you will need to install them with your system's package manager.

For Red Hat-based systems and other systems using Yum as the package manager, use the following command:

```
[root@local ~]# yum install libevent-devel libxml2-devel
```

For Ubuntu, Debian, and other systems that use `apt-get` or `aptitude`, the command is as follows:

```
[root@local ~]# aptitude install libxml2-dev libevent-dev
```

Building PHP

Once you have installed all the dependencies, you may start building PHP. Similar to other applications and libraries that were previously installed, you will basically need three commands: `configure`, `make`, and `make install`. Be aware that this will install a new instance of the application. If you already have PHP set up on your system, the new instance will not override it. Instead, it will be installed in a different location that is revealed to you during the `make install` command execution.

The first step (`configure`) is critical here, as you will need to enable the PHP-FPM options in order for PHP to include the required functionality. There is a great variety of configuration arguments that you can pass to the `configure` command; some are necessary to enable important features such as database interaction, regular expressions, file compression support, web server integration, and so on. All the possible configuration options are listed when you run this command:

```
[user@local php-5.6.10]$ ./configure --help
```

A minimal command may also be used, but be aware that a great deal of features will be missing. If you wish to include other components, additional dependencies may be needed, which are not documented here. In all cases, the `--enable-fpm` switch should be included:

```
[user@local php-5.6.10]$ ./configure --enable-fpm […]
```

The next step is to build the application and install it at the same time:

```
[user@local php-5.6.10]$ make && make install
```

This process may take a while, depending on your system specifications. Take good note of (some of) the information given to you during the build process. If you did not specify the location of the compiled binaries and configuration files, they will be revealed to you at the end of this step.

Post-install configuration

Begin by configuring your newly installed PHP, for example, copying the `php.ini` of your previous setup over to the new one:

> Due to the way Nginx forwards the script file and request information to PHP, a security breach might be caused by the use of the `cgi.fix_pathinfo=1` configuration option. It is highly recommended that you set this option to `0` in your `php.ini` file. For more information about this particular security issue, please consult the following article:
>
> `http://cnedelcu.blogspot.com/2010/05/nginx-php-via-fastcgi-important.html`

The next step is to configure PHP-FPM. Open up the `php-fpm.conf` file, which is located in `/usr/local/php/etc/` by default. We cannot detail all the aspects of the PHP-FPM configuration here (they are largely documented in the configuration file itself anyway), but there are important configuration directives that you shouldn't miss:

- Edit the user(s) and group(s) used by the worker processes and, optionally, the UNIX sockets
- The address(es) and port(s) on which PHP-FPM will be listening
- The number of simultaneous requests that will be served
- The IP address(es) allowed to connect to PHP-FPM

Running and controlling

Once you have made the appropriate changes to the PHP-FPM configuration file, you may start it with the following command (the file paths may vary depending on your build configuration):

```
[user@local ~]# /usr/local/php/sbin/php-fpm -c /usr/local/php/etc/php.ini
--pid /var/run/php-fpm.pid --fpm-config=/usr/local/php/etc/php-fpm.conf
-D
```

The preceding command includes several important arguments:

- `-c /usr/local/php/etc/php.ini` sets the path of the PHP configuration file
- `--pid /var/run/php-fpm.pid` sets the path of the `PID` file, which can be useful for controlling the process via an init script
- `--fpm-config=/usr/local/php/etc/php-fpm.conf` forces PHP-FPM to use the specified configuration file
- `-D` *daemonizes* PHP-FPM (ensures it runs in the background)

Other command-line arguments can be obtained by running `php-fpm -h`:

PHP-FPM can be stopped via the `kill` or `killall` commands. Alternatively, you may use an init script to start and stop the process, provided the version of PHP you installed came with one.

Nginx configuration

If you have managed to configure and start PHP-FPM correctly, you are ready to tweak your Nginx configuration file to establish the connection between both parties. The following server block is a simple, valid template on which you can base your own website configuration:

```
server {
    server_name .website.com; # server name, accepting www
    listen 80; # listen on port 80
    root /home/website/www; # our root document path
    index index.php; # default request filename: index.php

    location ~* \.php$ { # for requests ending with .php
        # specify the listening address and port that you configured
previously
        fastcgi_pass 127.0.0.1:9000;
        # the document path to be passed to PHP-FPM
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_
name;
        # the script filename to be passed to PHP-FPM
        fastcgi_param PATH_INFO $fastcgi_script_name;
        # include other FastCGI related configuration settings
        include fastcgi_params;
    }
}
```

After saving the configuration file, reload Nginx using one of the following commands:

```
/usr/local/nginx/sbin/nginx -s reload
```

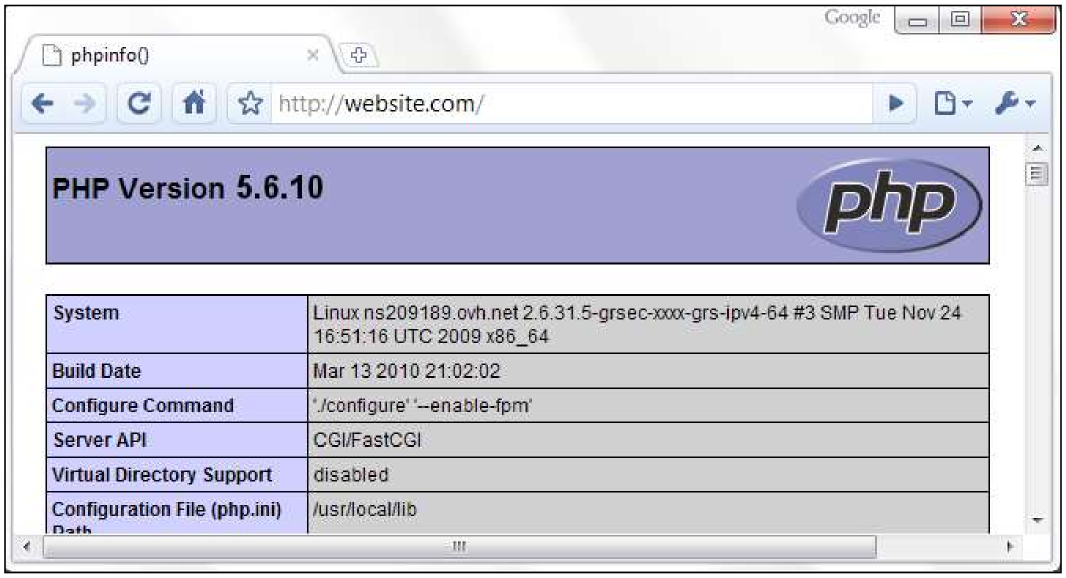
Or:

```
service nginx reload
```

Create a simple script at the root of your website to make sure that PHP is being interpreted correctly:

```
[user@local ~]# echo "<?php phpinfo(); ?>" >/home/website/www/index.php
```

Fire up your favorite web browser and load `http://localhost/` (or your website URL). You will see something similar to the following screenshot, which is the PHP server information page:



PHP Version 5.6.10

System	Linux ns209189.ovh.net 2.6.31.5-grsec-xxxx-grs-ipv4-64 #3 SMP Tue Nov 24 16:51:16 UTC 2009 x86_64
Build Date	Mar 13 2010 21:02:02
Configure Command	'./configure' '--enable-fpm'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib

Note that you may run into the occasional `403 Forbidden` HTTP error if the file and directory access permissions are not properly configured. If that is the case, make sure that you specified the correct user and group in the `php-fpm.conf` file, and that the directory and files are readable by PHP.

Python and Nginx

Python is a popular object-oriented programming language available on many platforms, from Unix-based systems to Windows. It is also available for Java and the Microsoft .NET platform. If you are interested in configuring Python to work with Nginx, it's likely that you already have a clear idea of what Python does. We are going to use Python as a server-side web programming language, with the help of the Django framework.